TRANSMITTAL OF APPEAL BRIEF

Docket No.
21776-00034-US

In re Application of: Hiroaki Kimura, et al.

Application No.	Filing Date	Examiner	Group Art Unit
09/241,735	February 2, 1999	F. Ferris	2123

Invention: APPARATUS FOR ANALYZING SOFTWARE AND METHOD OF THE SAME

TO THE COMMISSIONER OF PATENTS:

Transmitted herewith in triplicate is the Appeal Brief in this application, with respect to the Notice of Appeal filed: August 8, 2002 .

The fee for filing this Appeal Brief is 320.00 .

[x] Large Entity [ ] Small Entity

[ ] A check in the amount of _____ is enclosed.

[x] Charge the amount of the fee to Deposit Account No. 22-0185 .
This sheet is submitted in duplicate.

[ ] Payment by credit card. Form PTO-2038 is attached.

[x] The Commissioner is hereby authorized to charge any additional fees that may be required or credit any overpayment to Deposit Account No. 22-0185 .
This sheet is submitted in duplicate.

Larry J. Hume

Dated: December 30, 2002

Larry J. Hume
Attorney Reg. No.: 44,163
CONNOLLY BOVE LODGE & HUTZ, LLP
1990 M Street, N.W., Suite 800
Washington, DC 20036-3425
(202) 331-7111

Docket No.: 21776-00034-US
(PATENT)

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of: Conf. No. 9109

Hiroaki Kimura, et al.

Application No.: 09/241,735 Group Art Unit: 2123

Filed: February 2, 1999 Examiner: F. Ferris

For: APPARATUS FOR ANALYZING SOFTWARE AND METHOD OF THE SAME

APPELLANT'S BRIEF

Commissioner for Patents
Attn: Board of Patent Appeals and Interferences
Washington, DC 20231

December 30, 2002

Dear Sir:

This brief is in furtherance of the Notice of Appeal, filed in this case on August 8, 2002. A three (3) month extension of time is requested in the accompanying petition.

The fees required under §1.17(f) and any required petition for extension of time for filing this brief and fees therefor, are dealt with in the accompanying TRANSMITTAL OF APPEAL BRIEF.

This brief is transmitted in triplicate.

This brief contains items under the following headings as required by 37 C.F.R. §1.192 and M.P.E.P. §1206:

I. Real Party In Interest
II Related Appeals and Interferences
III. Status of Claims
IV. Status of Amendments
V. Summary of Invention
VI. Issues

VII. Grouping of Claims
VIII. Arguments
IX. Claims Involved in the Appeal
App. A Claims

I. REAL PARTY IN INTEREST

The real party in interest for this appeal is NS Solutions Corporation, Tokyo, Japan

II. RELATED APPEALS AND INTERFERENCES

There are no other appeals or interferences which will directly affect or be directly affected by, or have any bearing on the Board's decision in this appeal.

III. STATUS OF CLAIMS

A. Total Number of Claims in Application: There are 37 claims pending.

B. Current Status of Claims

1. Claims canceled: none

2. Claims withdrawn from consideration but not canceled: none

3. Claims pending: 1-37

4. Claims allowed: none

5. Claims rejected: 1-37

C. Claims On Appeal: The claims on appeal are claims 1-37

IV. STATUS OF AMENDMENTS

Applicants filed an Amendment in this case under 37 C.F.R. §1.111 on April 8, 2002, in response to the non-final Official Action dated November 7, 2001. The Examiner responded to the Amendment with a Final Rejection mailed May 8, 2002 In the Final Official Action, the Examiner indicated that Claims 1-37 were finally rejected.

Accordingly, the claims enclosed herein as Appendix A incorporate the amendments indicated in the paper filed by Applicant on April 8, 2002.

V. SUMMARY OF THE INVENTION

The disclosed and claimed invention relates generally to a software analysis apparatus, a software analysis method, and a computer-readable medium recording a computer program for making a computer implement functions related to analyzing a computer program to help the user easily understand the contents of the computer program and the complex interactions within the program. More specifically, the disclosed and claimed invention relates to a software analysis apparatus, method, and computer-readable medium for analyzing conventionally obtained program analysis information in a novel and non-obvious way.

Although the disclosed and claimed apparatus, method, and computer-readable medium are submitted as being novel and non-obvious, ***the actual program analysis information which may be processed by the invention is submitted as being known in the art***, and may include information obtained from conventional software program analysis techniques such as, for example, graphically displayed call graph and flow graph, program influence range analysis, program structure analysis, and program data flow anomaly analysis.

According to disclosed and claimed embodiments of the invention, conventionally generated program analysis information generated by program analysis information generation means is stored in a data recording medium in arbitrary units or at an arbitrary timing, and preferably is stored in the form of an object-oriented database. Thus, program analysis information from a large scale program may be safely and reliably obtained and retrieved without experiencing the conventional problems of a memory shortfall in the middle of analysis, even when the memory capacity mounted on the computer or analysis workstation is limited.

Even when the analysis process is interrupted due to some computer or operator-related problem, e.g., memory shortage, information stored in the data recording medium, at least to the point of occurrence of the problem, is safely protected. Hence, contrary to conventional approaches, the analysis process is able to be resumed from the point where the process was interrupted, thus avoiding repetitive processing, greatly improving analysis efficiency, and reducing operator effort and time required to complete the analysis.

When an analysis process is to be interactively accomplished after generation of conventional analysis information, and since analysis information already stored in the database can be directly used, the time required for generating that analysis information can be omitted, and the wait time of the operator can be reduced compared to a conventional system which generates analysis information from the beginning, every time new analysis process is started due to a previously occurring technical problem which cut short the original analysis before processing had completed.

A. Background Discussion of Conventional Software Analysis Information

To set the stage for this appeal, and in light of the Examiner's comments and apparent misunderstanding of the novel and non-obvious aspects of the claimed invention, a somewhat longer background discussion of conventionally-available software analysis information is submitted as being necessary to a better appreciation of the claims on appeal.

By way of background, when a large number of programmers develop a large-scale program in collaboration, or maintain a software program already in existence, it is often difficult for a given programmer to understand the program code written by other programmers. The disclosed and claimed invention is directed to not only providing various kinds of program analysis information, such as call graph and flow graph, and automatically analyzing program source code, in order to help understanding of the program, especially for analyzing a large-scale program, but is also directed to a novel and non-obvious way to process the conventionally generated program analysis information.

When a large-scale program is to be analyzed, e.g., millions of lines of code or program steps, the volume of analysis information required for program analysis also becomes very large, oftentimes resulting in poor analysis efficiency. For example, the maximum memory capacity in an analysis computer or workstation may be insufficient for all pieces of analysis information, or the processing time required for analysis may become very long.

When memory capacity is insufficient to accommodate the quantity of program analysis information generated, further analysis of remaining program steps is not possible. In such circumstances, the program to be analyzed is segmented or re-formed into various pieces.

However, the program must again be analyzed again from the beginning, resulting in poor analysis efficiency.

If program segmentation is used to facilitate analysis in such a case, certain types of analysis, e.g., influence analysis which detects the influence of a change in value of a given variable in the program on other parts of the program, cannot be relied upon to yield appropriate and truly representative analysis information which relates to the interactions of the overall program, because of the interruption of the process due to program segmentation.

When the processing time becomes too long, processing may have to be interrupted for higher priority processing tasks, or for other considerations, such as operator fatigue, end of shift, etc. When processing is resumed, the analysis processing in many cases will have to start from the beginning, resulting in inefficiencies, poor analysis efficiency, and further expense.

In addition, when the maximum memory capacity (main storage device used as a work memory) that a normal workstation or personal computer can mount is insufficient for all pieces of analysis information, some serious problems are likely to occur.

For example, when the memory capacity becomes inadequate during analysis, and no more analysis information can be stored in a memory, the remaining analysis for the program cannot be done. In such case, after some modifications are made, e.g., the program to be analyzed is segmented or re-formed into some pieces, the program must be analyzed from the beginning, resulting in poor analysis efficiency.

Furthermore, analysis may be interrupted due to some cause other than memory shortage during program analysis. In such a case, all analysis obtained so far cannot be used, and program analysis must be redone from the beginning, resulting in poor analysis efficiency.

Examples of conventionally available program analysis information are discussed below.

A "call graph", for example, indicates a calling relationship among procedures (e.g., "functions" in C) that comprise a program. Nodes of this type of graph represent procedures, and edges represent the calling relationship among the procedures. The call graph shows how much time was spent in each function and its children. From this information, you can find

functions that, while they themselves may not have used much time, called other functions that did use unusual amounts of time. When visually displayed, the program analyst or programmer not only can visually recognize the calling relationship among the procedures, but can also easily understand the structured level and contents of a program, or can easily detect an unwanted procedure call.

In contrast, a "flow graph" shows the flow of control in a given procedure. Nodes in this type of graph represent fundamental blocks, and edges represent the flow of control among the fundamental blocks. When this is information is graphically displayed, the control flow in that particular procedure can be visually recognized, and unwanted control flow can be easily detected. For example, "dead code" that control cannot reach is a node which cannot be reached even if the control follows the edges from an entrance node. However, such dead code can easily be detected, since it does not have any link from the entrance node.

A "symbol table" includes a group of information that represent meanings of symbols (variables) used in a program. For example, when the character "a" is used at different locations in a program, they may indicate identical variables or different variables. A symbol table provides a base for the coalescence of data relating to the various elements of the source text and provides a possibility for describing certain relationships between the text and the target machine, such as the assigned, and possibly relative, address of variables. For example, when variables expressed by an identical character are used in two different functions, if those variable are global variables, they are identical variables; and if the variables are locally defined in the individual functions, they represent different variables. A symbol table can provide necessary data for the acquisition of library-provided subprograms, and may also, during run time, act as a transfer vector for the linkage of the generated code and those subprograms.

Also, a "syntactic analysis tree" expresses a program configuration using a tree structure. Upon analyzing a program, syntactic analysis is generally done first by expressing a source program in the form of a tree structure so that the computer can easily understand the program.

The "data flow information" is information resulting from analysis of the data flow such as alias information of a pointer, or definition use chain information that indicates the use

location of a variable defined at a given location, for example. Using this data flow information, any data flow anomalies can be inspected.

A "data flow anomaly" is a combination of illegal events with respect to data. All data must be used while observing specified rules, i.e., each data must be used after it is defined, and may not be used after it is finally undefined. An illegal combination that does not abide by such rules causes a data flow anomaly. Even when data flow anomaly is present, source code can be compiled. However, upon executing the compiled program in practice, a data flow anomaly may appear when control takes an anomalous path. Hence, such an anomaly is preferably removed in advance in the processing of source code.

A "program dependence graph" expresses control dependence in a program (dependence among functions), and data dependence (when data are substituted in functions) in the form of a graph. When the program dependence graph is generated, for example, influence range analysis of a program can be executed. The influence range analysis analyzes the range of influence imposed when a certain location (one sentence or one variable, for example) in a program has been changed.

"Module I/O information" pertains to input/output ("I/O") of modules (e.g., functions in C) that form a program. This information describes I/O information using I/O functions and global variables, in addition to I/Os as arguments and return values.

"Metrics information" pertains to numeration indices of software. Metrics which represent quantitative complexity, and those which represent qualitative complexity may be generated. As metrics of quantitative complexity, two different kinds of quantities, i.e., a size metric that measures the physical description quantity of a program, and cyclomatic number that measures the complexity of a control structure are measured. On the other hand, as metrics of qualitative complexity, cohesion and coupling that express the contents of modules are often measured.

"Redundancy information" pertains to a redundant sentence that does not influence output in a single procedure. Since a redundant sentence does not influence output, if it is deleted from the source code, the external output remains unchanged. Hence, by confirming

such redundant sentences in accordance with this redundancy information, and taking measures against them, unintended operation can be prevented.

"Maintenance document information" refers to a document group used upon maintaining a program. More specifically, this information describes lists of procedures, types, variable names defined in a program, for example, and how the individual procedures are related. This "maintenance document information" may be conventionally used when a given programmer wants to understand a program created by another programmer. By acquiring the "maintenance document information", information can be provided in the hypertext format, and is far more convenient as compared to paper-based documents.

These are non-exhaustive examples of conventional program analysis information commonly used and understood in a software development or software maintenance environment.

Applicants submit that it is common knowledge to persons having skill in the art that metrics information, redundancy information and maintenance document information relating to a software program or system may be generated on the basis of program analysis information obtained from analysis of the program source code, syntactic analysis tree, symbol table, call graph, flow graph, data flow information, program dependence graph, and module I/O information, as the above examples describe.

Further, although software program metrics information, redundancy information, and maintenance document information relating to a software program may be conventionally generated as a result of a batch process, the specific methods actually used to generate such conventionally generated information are submitted as not being essential to an understanding of the disclosed and claimed invention, except as necessary to provide a general background and contextual understanding of the area of computer and software technology impacted by Applicants' invention.

What conventionally known software program analysis devices and techniques lack is a way to handle such types of conventionally obtained analysis information generated from large scale programs, without making large demands upon memory, taking longer than desired to

process, or requiring program segmentation which may adversely impact the results of analysis information obtained from overall program interactions, and which consequently do not accurately represent the true program interactions. What conventionally known software program analysis devices and techniques also lack is a way to compensate for a failure of a previous process to complete, without having to start the analysis completely over.

B. Discussion of Various and Preferred Embodiments

The disclosed and claimed invention has been developed in consideration of the above situation, and has, as one objective, efficient analysis of large-scale programs, and to provide a software analysis apparatus and a software analysis method that allows efficient use of the conventionally obtained analysis information, even when analysis is interrupted due for some reason during program analysis.

The disclosed and claimed invention finds particular utility in the analysis of relatively large-scale software programs, by processing conventional program analysis information identified as above, for example, in a new, novel, and non-obvious manner, as discussed below, and as disclosed in detail in Applicants' originally-filed disclosure.

In one embodiment of the invention, program analysis information is sequentially stored in a predetermined data recording medium in an arbitrary unit or at an arbitrary timing, as variously claimed in the pending claims.

This approach allows this embodiment of the invention to achieve considerable cost and time-saving effects, including safely obtaining program analysis information without causing any memory shortage in the middle of analysis, even when the memory capacity mounted on the computer is limited. Further, even when the analysis process is interrupted due to some technical or operator problem, information stored in the data-recording medium up until the time of interruption is safely protected. Hence, the process is resumed from the point at which it was interrupted, thus avoiding repetitive processing, and greatly improving analysis efficiency, reducing time for complete analysis, and reducing the resultant cost of analysis.

Furthermore, when interactive analysis processing is done after the generation of analysis information, since analysis information already stored in the database can be directly used, the

time required for generating the previously obtained analysis information can be omitted, so that the wait time of the operator can be reduced, as compared to a conventional analysis system which generates analysis information from the beginning, every time a new analysis process is started.

Turning now to Fig. 1 of the Drawings, in a preferred embodiment of the invention, source file 10 records source code of a program written in a predetermined language (e.g., C). Source file 10 is loaded into program analysis information generation unit 11. Program analysis information generation unit 11 executes a process for generating conventional program analysis information, e.g., a known process as discussed above, required for program analysis in accordance with user's instructions received by various methods, e.g., interactive inputs with a computer, and/or inputs of setup files (See Specification at p. 7, line 24, through p. 8, line 18).

Program analysis information storage unit 12 sequentially stores program analysis information in a database on an information recording medium 13 such as a hard disk when program analysis information generation unit 11 generates program analysis information. The generated program analysis information may include one or more of the conventionally obtained types of analysis information, as discussed in the preceding subsection of this Brief. More specifically, whenever program analysis information is generated, program analysis information generation unit 11 informs program information storage unit 12 by generation of a message, for example. Upon receiving the message, program analysis information storage unit 12 stores the program analysis information generated at that time on information recording medium 13 (See Specification at p. 8, lines 6-18).

Upon executing a program analysis process, program analysis information read unit 14 preferably reads out program analysis information required for program analysis processing unit 15 from information recording medium 13, and supplies such information to program analysis processing unit 15 (See Specification at p. 8, lines 19-24).

When the memory in an analysis computer or workstation required to store program analysis information from previous program analysis is insufficient, or when generation of analysis information is interrupted after some pieces of program analysis information have already been stored in information recording medium 13, program analysis information read unit

14 reads out information indicating that the partial information has already been stored in information recording medium 13, and the program analysis information generation unit 11 generates missing program analysis information based on the readout information which provides knowledge of the already obtained information (See Specification at p. 8, line 24, through p. 9, line 8).

Program analysis information generation unit 11, program analysis information storage unit 12, program analysis information read unit 14, and program analysis processing unit 15 are preferably implemented by a computer including a CPU, ROM, and RAM, for example. The computer preferably includes a main storage device, e.g., a memory for temporarily storing program analysis information generated by program analysis information generation unit 11, before the information is finally transferred to information recording medium 13 (See Specification at p. 9, lines 9-19).

In this embodiment, program analysis information may include, for example, one or more of conventionally generated syntactic analysis tree (including a symbol table), call graph, flow graph, data flow information, program dependence graph, module I/O information, metrics information, redundancy information, or maintenance document information, as shown in Fig. 2. Of these types of analysis information, the syntactic analysis tree, call graph, flow graph, data flow information, program dependence graph, module I/O information, are preferably generated by program analysis information generation unit 11 shown in Fig. 1, while metrics information, redundancy information, maintenance document information, for example, are preferably generated by program analysis unit 20_i shown in Fig. 3, which may analyze a program by way of a batch process, for example (See Specification at p. 9, line 20, through p. 10, line 6).

In a preferred embodiment, program analysis information generation unit 11 shown in Fig. 1 sequentially generates various kinds of program analysis information mentioned above, using a hierarchical structure, as shown in Fig. 2. The reason why such a hierarchical structure is preferred, is to be able to use program analysis information in lower hierarchical layers when program analysis information in an upper layer is generated. Upon completion of generation of a group of program analysis information for each layer, the program analysis information storage unit 12 sequentially stores these pieces of analysis information in the database as "transactions", by way of a "transaction function" (See Specification at p. 14, line 21, through p. 15, line 4).

For example, in order to generate a call graph or flow graph, information of the syntactic analysis tree of the lower layers is required. Also, in order to generate data flow information, information of the syntactic analysis tree of the lower layers, and information of a call graph and flow graph are required. Inspecting for data flow anomaly by an interactive process using program analysis processing unit 15, for example, requires various kinds of program analysis information, hierarchically shown in Fig. 2, such as syntactic analysis tree, call graph, flow graph, and data flow information (See Specification at p. 15, lines 5-16).

In this way, after generation of given program analysis information in an upper layer, a hierarchical relationship is established that requires program analysis information in lower layers, which has already been generated and stored in information recording medium 13. In one embodiment, program analysis information generation unit 11 sequentially generates program analysis information for lower layers shown in Fig. 2, and program analysis information storage unit 12 sequentially preferably stores the obtained analysis information in information recording medium 13. Then, program analysis information read unit 14 , reading from information recording medium 13, reads out the program analysis information in a lower layer which may be required upon generating given program analysis information in an upper layer, and supplies it to program analysis information generation unit 11 (See Specification at p. 15, line 17, through p. 16, line 6).

As previously mentioned, especially in the case of a large scale program, if all pieces of program analysis information are conventionally generated by a batch process and conventionally processed without using a "transaction" or "transaction function" as in the disclosed invention as discussed above, and if the program analysis information is not then periodically stored in information recording medium 13; the volume of data pertaining to source code in source file 10 to be analyzed is most likely to exceed the memory capacity that can be mounted on a workstation or personal computer, and continued analysis is likely to become difficult or impossible, thus wasting the processing results previously obtained up until the point that memory capacity was exceeded (See Specification at p. 16, lines 7-16).

In contrast, in the software analysis apparatus of this embodiment, the program analysis information storage unit 12 classifies program analysis information generated by the program analysis information generation unit 11 in units of types or kinds of analysis information present

in the respective layers (see, e.g., Fig. 2), and preferably sequentially stores such classified information in a database format on information recording medium 13, which may be a hard disk, or other non-volatile storage medium (See Specification at p. 16, lines 17-24).

In this way, the problems associated with the limited memory capacity of the computer or workstation essentially become irrelevant. Hence, upon analyzing an especially large scale program, program analysis rarely encounters memory shortage, and analysis can be reliably done to completion. Since analysis information in each layer in the data structure shown in Fig. 2 is stored as a transaction in information recording medium 13 every time it is generated, the various kinds of program analysis information already stored in information recording medium 13 are safely protected, even when some trouble other than memory shortage is encountered (See Specification at p. 16, line 25 through p. 17, line 3).

For example, if a computer goes down due to some technical problem in the middle of a long analysis when a large scale program being analyzed requires several days between generation of a syntactic analysis tree until generation of a program dependence graph, for example, and if data mapped on the memory are destroyed, at least data already stored in information recording medium 13 in the form of a hierarchical database can be safely protected. Then, when analysis is restarted, the program analysis information already stored in the database need not be generated again, and the analysis can be restarted at a point for which process information has not yet been generated, usually, for example, at a point designated by the operator. In this way, according to this embodiment, conventional program analysis information that has been obtained before interruption can be effectively retained and used, and repetitive operations can be avoided (See Specification at p. 17, line 12 through p. 18, line 2).

In a preferred embodiment, an object oriented database and software are used, and program analysis information may be generated using an object oriented language, e.g., C++. Such a database is preferred, since program analysis information may have a complicated structure, and relations such as a graph structure, for example, if a relational database is used, make it difficult to store information. Also, such an object-oriented database is more convenient, since program analysis information generated as an object in C++ or other object-oriented language can readily be stored in the hierarchical structure as the information is generated (See Specification at p. 18, lines 3-13).

The internal arrangement and operation of program analysis processing unit 15 will be discussed with reference to Fig. 3. Fig. 3 shows a plurality of program analysis units 20_1, 20_2,... 20_i, generally referred to as program analysis units 20. Program analysis units 20 execute various kinds of conventional program analysis on the basis of the source code of the source file 10, and program analysis information is stored in the database in the information recording medium 13 by program analysis information storage unit 12 (See Specification at p. 18, lines 14-25).

Program analysis units 20_1, 20_2,... 20_i provide the various kinds of conventional analysis processes being undertaken. In this embodiment, for example, program analysis unit 20_1 may execute an analysis process for graphically displaying a call graph on the computer screen. With this process, the programmer can visually confirm the contents of the call graph, and easily understand the calling relationship among functions in the program. In this case, program analysis unit 20_1 loads source code, and reads out information of the syntactic analysis tree and call graph as program analysis information from the information recording medium 13 via the program analysis information read unit 14, shown in Fig. 1 (See Specification at p. 18, line 26 through p. 19, line 11).

Program analysis unit 20_2 may execute an analysis process for graphically displaying a flow graph on the computer screen. With this process, the programmer can visually confirm the contents of the flow graph, in practice, and can easily understand the flows of variables in a given function. In such a situation, program analysis unit 20_2 loads source code, and reads out information of the syntactic analysis tree and flow graph as program analysis information from information recording medium 13 via the program analysis information read unit 14 shown in Fig. 1 (See Specification at p. 19, lines 11-21).

Program analysis unit 20_i may execute an analysis process that can analyze using a batch process relying upon various kinds of program analysis information already stored in information recording medium 13 by program analysis information storage unit 12, and which may store the process results as a batch analysis results file in information recording medium 13. During this batch process, the operator preferably need not input any instructions, and the apparatus of this embodiment preferably automatically analyzes the information. The resulting batch analysis result file may include, for example, metrics information pertaining to the program scale,

redundancy information, maintenance document information, for example . (See Specification at p. 19, line 22 through p. 20, line 6).

In this embodiment, these program analysis units 20_1, 20_2,..., 20_i fetch prestored program analysis information from information recording medium 13, execute further conventional analysis processes, and display analysis results 21_1, 21_2,..., 21_i on the computer screen using GUIs (Graphical User Interfaces) 22_1, 22_2,..., 22_i, which preferably display the analysis process results so that the operator can observe the results on a computer screen, for example (See Specification at p. 20, lines 17-25).

Thereafter, when the operator wants to input instructions upon observing the displayed results, he or she interactively inputs instructions to the program analysis units 20_1, 20_2,... via the GUIs 22_1, 22_2,... to make the program analysis units 20_1, 20_2,... redo analysis processes using the prestored information from information recording medium 13, and display their results. The program analysis progresses while repeating such processes (See Specification at p. 20, line 26 through p. 21, line 11).

Other examples of conventional analysis processes executed by program analysis units 20 include influence range analysis, structure analysis, and data flow anomaly analysis, for example. In one embodiment, the data flow anomaly analysis is preferably done by an interactive process, but may be implemented by a batch process (See Specification at p. 21, line 12 through p. 22, line 8).

The overall operation of the software analysis apparatus of a preferred embodiment will be explained with reference to the flowchart in Fig. 4, and the Specification at p. 23, line 1 through p. 24, line 17. If the operator designates source code to be analyzed in step S1 of Fig. 4, step S2 checks to see if the database containing program analysis information corresponding to the designated source code has already been stored in information recording medium 13. If such a database is found, then program analysis information read unit 14 reads out the program analysis information in step S3, rather than requiring regeneration of the information.

In step S4, the operator instructs the software analysis apparatus to generate conventional program analysis information. For example, the operator instructs the range of analysis

information to be generated, in sequential order from the lowermost layer. If the process was interrupted due to some problem in the middle of the previous generation process, and if that generation process must be restarted to provide needed data, the operator provides instruction concerning the analysis information for which generation is to be restarted.

In step S5, program analysis information generation unit 11 generates program analysis information in accordance with analysis information generation instructions input by the operator, for example. At this time, as the analysis process progresses, the generated program analysis information is sequentially stored, temporarily, in the memory. In step S6, the program analysis information storage unit 12 transfers and stores the conventionally generated program analysis information from the memory to information recording medium 13 in the form of a database. The analysis information is preferably also classified, e.g., into one of the hierarchical types in the database structure shown in Fig. 2.

Upon complete generation of required program analysis information, the operator may preferably instruct the apparatus of any program analysis processes to be executed in step S7. In response to this instruction, program analysis processing unit 15 executes program analysis processes using corresponding ones of the program analysis units 20_1, 20_2,..., 20_i shown in Fig. 3 (step S8).

Upon completion of these program analysis processes, a decision is made in step S9 whether another kind of program analysis is to be done. If the decision in step S9 is "YES", flow then returns to step S7; otherwise, flow advances to step S10. A decision is made in step S10 whether additional source code is to be analyzed. If the decision is "YES" in step S10, flow returns to step S1; otherwise, the software analysis process ends.

Implementation of a computer readable recording medium embodying the functionality of the disclosed apparatus and method described above is believed to be readily discernible from the above discussion and disclosure in the Specification. The software analysis apparatus of a preferred embodiment is implemented when the computer operates in accordance with a program stored in its ROM, RAM, for example, to carry out functions associated with a software analysis process, including storing program analysis information in a database, and requesting

regeneration of such information when necessary due to some problem where the original analysis was interrupted, and/or the data is incomplete.

The invention may be varied from the above embodiments. For example, in a preferred embodiment above, at the time when one type or kind of analysis information is generated in each layer of the hierarchical data storage layers, the program analysis information is stored in a database. Alternatively, when generation of various types of analysis information in each layer is completed, or when a predetermined volume of information is stored in the computer memory, such information may then be transferred to and stored in a database. Also, after a predetermined period of time has elapsed, the program analysis information obtained in that time period may be stored in a database.

To reiterate, according to disclosed embodiments of the invention, since conventionally generated program analysis information generated by the program analysis information generation means is stored in a predetermined data recording medium in arbitrary units or at an arbitrary timing, program analysis information obtained from a large scale program may be safely and reliably obtained without experiencing any memory shortage in the middle of analysis, even when the memory capacity mounted on the computer or analysis workstation is limited. Even when the analysis process is interrupted due for any reason, information already stored in the data recording medium so far is safely protected. Hence, the process is resumed from the point of the interrupted process, thus avoiding repetitive processes, and greatly improving analysis efficiency, and reducing operator effort and time.

When some analysis process is to be interactively done after generation of analysis information, since analysis information already stored in the database can be directly used, the time required for generating that analysis information can be omitted, and the wait time of the operator can be reduced compared to a conventional system which generates analysis information from the beginning every time new analysis process is started.

VI. ISSUES

A. Is the Examiner's objection to the Specification reasonable merely because non-essential, background information discussed with respect to referenced foreign applications in the "Background of the Invention" section, specifically the

"Description of Related Art" subsection, have not been translated into English and physically embodied into the present Specification?

B. Has the Examiner established that the Specification lacks support for conventional aspects of claims 1-37 under 35 U.S.C. §112, first paragraph, given the general knowledge and skill available to a person having skill in the art, in light of known, conventional approaches at the time of Applicants' invention, especially given the Examiner's admission that such aspects were actually conventional and known at that time?

C. Has the Examiner established that every recited feature of claims 1-6, 8-10, and 35 is disclosed by Wygodny et al. (US 6,282,701), and therefore anticipated under 35 U.S.C. §102(e)?

D. Has the Examiner established that every recited feature of claims 13-17, 19-21, and 36 is disclosed by Wygodny et al. (US 6,282,701), and therefore anticipated under 35 U.S.C. §102(e)?

E. Has the Examiner established that every recited feature of claims 24-28, 30-32, and 37 is disclosed by Wygodny et al. (US 6,282,701), and therefore anticipated under 35 U.S.C. §102(e)?

F. Has the Examiner established that every recited feature of claims 7, 11, 12, 18, 22, 23, 29, 33, and 34 is disclosed by Wygodny et al. (US 6,282,701), and therefore anticipated under 35 U.S.C. §102(e)?

VII. GROUPING OF CLAIMS

For purposes of this appeal brief only, and without conceding the teachings of any prior art reference, the claims have been grouped as indicated below:

Group	Claims
I	1-6, 8-10, and 35
II	13-17, 19-21, and 36
III	24-28, 30-32, and 37
IV	7, 11, 12, 18, 22, 23, 29, 33, and 34

In Section VIII below, Applicant has included separate arguments supporting the separate patentability of each claim group as required by M.P.E.P. §1206.

VIII. ARGUMENTS

A. Is the Examiner's objection to the Specification reasonable merely because non-essential, background information discussed with respect to referenced foreign applications in the "Background of the Invention" section, specifically the "Description of Related Art" subsection, have not been translated into English and physically embodied into the present Specification?

Appellants submit that the Examiner's objection to the Specification is not reasonable, because the foreign applications mentioned briefly in the Background section are merely indicative of the conventional state of the art, and are not necessary to a proper understanding of the claimed invention.

Appellants point out that "essential material" is defined as that which is necessary to (1) describe the claimed invention, (2) provide an enabling disclosure of the claimed invention, or (3) describe the best mode (35 U.S.C. 112). In any application which is to issue as a U.S. patent, essential material may not be incorporated by reference to (1) patents or applications published by foreign countries or a regional patent office, (2) non-patent publications, (3) a U.S. patent or application which itself incorporates "essential material" by reference, or (4) a foreign application.

However, ***nonessential*** subject matter may be incorporated by reference to (1) patents or applications published by the United States or foreign countries or regional patent offices, (2) prior filed, commonly owned U.S. applications, or (3) non-patent publications.[1] Nonessential subject matter is subject matter referred to for purposes of indicating the background of the invention, or illustrating the state of the art. Appellants submit that this is the case in this Appeal.

The Examiner objected that Appellants' background Japanese references (i.e., JP 9-32415 and JP 9-32452) must be translated into English, and incorporated into the disclosure. In addition, the Examiner indicated that any translation must be accompanied by an affidavit or declaration executed by the Applicant or Applicant's Representative stating that the material added to the Specification consists of the same material incorporated by reference in the

[1] *See* MPEP §608.01.

application.

Appellants submit that none of their cited Japanese applications, mentioned in passing in the Background Section, are "essential material" which must be incorporated bodily into the Specification by translation and amendment. These references have been mentioned only to apprise the reader of some of the conventional program analysis approaches available prior to this application (See Specification at p. 2, line 27 -28). Appellants' cited Japanese applications are also mentioned only briefly in the Disclosure with respect to indicating known methods of detection of "data flow anomaly" (See Specification at p. 12, lines 25-28). Appellants submit that this aspect of conventional techniques should also not be considered "essential material" for the purposes of this application.

These references have been referred to in the Specification only in an attempt to indicated, in passing, some of the available conventional program analysis techniques or tools, and are submitted as being "non-essential".

Although Appellants submit that they should not be required to endure further expense and delay in prosecuting this Appeal, in the event that the Honorable Board believes that such conventional, background material objected to, in its absence, by the Examiner, is, indeed, "essential material", Appellants will arrange to translate the Japanese references, and submit an amendment which bodily incorporates the translated references into the Background section of the Disclosure.

Reversal of the Examiner's objection to the Specification is requested.

B. Has the Examiner established that the Specification lacks support for conventional aspects of claims 1-37 under 35 U.S.C. §112, first paragraph, given the general knowledge and skill available to a person having skill in the art, in light of known, conventional approaches at the time of Applicants' invention, especially given the Examiner's admission that such aspects were actually conventional and known at that time?

The Examiner has not established that the Specification lacks support for the conventional aspects of claims 1-37, and reversal of the enablement rejections is therefore requested.

The Examiner rejected claims 1-37 under 35 U.S.C. §112, first paragraph, as not being enabled by the Specification, as discussed above with respect to the objection to the Specification as not containing translations of Appellants' background Japanese references. The Examiner asserts that the present application does not specifically explain the "analysis process" used.

The Examiner indicates that no algorithms are given and no analytical process is described, and that the specification makes reference to the claimed invention analyzing a computer program, and automatically generating program analysis information, but does not specifically explain the process. However, in the specification of the present invention, there is a description of the processes executed by the program analysis means, such as call graph and flow graph which are graphically displayed, influence range analysis, structure analysis and data flow anomaly analysis.

Appellants submit, as discussed in the Summary of the Invention section of this Brief, that the thrust of Applicants' invention is not the analysis process, *per se*, but rather how information generated by conventional analysis processes is used and analyzed in a more efficient manner. Appellants submit that a person having skill in the art would know the various conventional "analysis processes" which could benefit from Applicants' invention.

Applicants submit that, since each of these conventional processes are generally known, the specification of the present invention is not required to explain these processes in detail. Although metrics information, redundancy information and maintenance document information are generated as a result of a batch process, the methods of generating information are also not essential material in the disclosure of the present invention.

In addition, Applicants submit that it is common knowledge to those with skill in the art that metrics information, redundancy information and maintenance document information are generated on the basis of program analysis information, such as source code, syntactic analysis tree, symbol table, call graph, flow graph, data flow information, program dependence graph, and module I/O information.

In fact, the Examiner admits "...that processes relating to source code, syntactic analysis tree, symbol table, call graph, flow graph, and data flow information are obvious and well

known...."[2] Appellants reiterate that generation of this type of information is not the invention; rather the claimed invention deals with how to manage such conventionally generated information.

The Examiner goes on to incorrectly assert that Appellants have claimed an apparatus and method for "program analysis" as a basis for all independent claims, and have not claimed the conventional types of information quoted above.[3]

Appellants point out that independent claims 1 and 11 claim a ***software analysis*** apparatus; independent claims 13 and 22 claim a ***software analysis*** method; and independent claims 24 and 33 claim a ***computer readable recording medium*** recording a computer program for making a computer implement unique and non-obvious functions relating to processing of conventionally generated program analysis information.

The Examiner bases his enablement rejection on various references in the Specification to "program analysis" and "analysis process", which in his estimation, are a critical part of the invention, and which are not disclosed in explicit detail. Appellants again reiterate that these conventionally available techniques are not required to be disclosed in the Specification, as the invention is not drawn to these processes, *per se*, but is directed to storage, handling, and post-processing of the program analysis information which is generated by known techniques.

Further, the Examiner asserts that processes such as "metrics information", "redundancy information", and "maintenance document information" do not have an equivalent specific meaning in the art, and are not sufficiently defined by the Specification. Appellants traverse this assertion, because these terms are submitted as being adequately defined in Appellants' Specification.

For example, Appellants submit that metrics information is known to be a measure of software quality which indicate the complexity, understandability, testability, description and intricacy of code. Further, Appellants, in their Specification, have provided definitions of each of these terms, as discussed in the background discussion of this Appeal Brief, and as repeated

---

[2] *See* Final Official Action, p. 7, paragraph 6, lines 11-13.
[3] *See* Final Official Action, p. 7, paragraph 6 through p. 8, line 2.

below with reference to the Specification.

"Metrics information" pertains to numeration indices of software. Metrics which represent quantitative complexity, and those which represent qualitative complexity may be generated. As metrics of quantitative complexity, two different kinds of quantities, i.e., a size metric that measures the physical description quantity of a program, and cyclomatic number that measures the complexity of a control structure may be measured. On the other hand, module cohesion and module coupling relating to the contents of modules are often measured, and are representative of the qualitative complexity of the software (See Specification, as previously amended, at p. 13, lines 15-27).

"Redundancy information" pertains to a redundant sentence that does not influence output in a single procedure. Since a redundant sentence does not influence output, if it is deleted from the source code, the external output remains unchanged. Hence, by confirming such redundant sentences in accordance with this redundancy information, and taking measures against them, unintended operation can be prevented (See Specification, at p. 14, lines 1-8).

"Maintenance document information" refers to a document group used upon maintaining a program. More specifically, this information describes lists of procedures, types, variable names defined in a program, for example, and how the individual procedures are related. This "maintenance document information" may be conventionally used when a given programmer wants to understand a program created by another programmer. By acquiring the "maintenance document information", information can be provided in the hypertext format, and is far more convenient, as compared to paper-based documents (See Specification, at p. 14, lines 9-20).

Accordingly, Appellants submit that the terms above are adequately defined or known in the art and, considering the knowledge of a person having skill in the art with respect to the known program analysis tools and techniques, a person having skill in the art can fully understand and appreciate the novel and non-obvious features of the recited invention from the description contained in the present specification.

Further, to the extent that the enablement rejection of the claims is related to the Examiner's objection to the Specification, discussed above with respect to incorporation of non-

essential material in Japanese references, Appellants point out the existence of case law that holds that mere reference to another application, patent, or publication is not an incorporation of anything therein into the application containing such reference, for the purpose of the disclosure required by 35 U.S.C. 112, first paragraph.[4]

Reversal of the Examiner's enablement rejection of claims 1-37 is therefore requested.

C. Has the Examiner established that every recited feature of claims 1-6, 8-10, and 35 in Group I is disclosed by Wygodny et al. (US 6,282,701), and therefore anticipated under 35 U.S.C. §102(e)?

Applicants note that anticipation requires the disclosure, in a prior art reference, of each and every limitation as set forth in the claims.[5] There must be no difference between the claimed invention and reference disclosure for an anticipation rejection under 35 U.S.C. §102.[6] To properly anticipate a claim, the reference must teach every element of the claim.[7] "A claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference".[8] "The identical invention must be shown in as complete detail as is contained in the ...claim."[9] In determining anticipation, no claim limitation may be ignored.[10]

In view of the foregoing authority, Wygodny et al. at least fails to anticipate independent claim 1, and also fails to anticipate dependent claims 2-6, 8-10, and 35, depending from claim 1.

With reference to the general differences in the claimed invention, and with particular respect to the recited distinctions of pending claims 1-6, 8-10, and 35 of the claimed invention over Wygodny et al., and considering the Examiner's comment that Wygodny et al. disclose a program analyzer method and apparatus having a GUI (Graphical User Interface) which collects trace information for use in analyzing a computer program, ***Applicants point out that, contrary to the Examiner's assertion, "trace", as used in Wygodny et al., and "analysis information" in the present invention are technically different concepts.***

---

[4] *In re de Seversky*, 474 F.2d 671, 177 USPQ 144 (CCPA 1973).
[5] *Titanium Metals Corp. v. Banner*, 227 USPQ 773 (Fed. Cir. 1985).
[6] *Scripps Clinic and Research Foundation v. Genentech, Inc.*, 18 USPQ2d 1001 (Fed. Cir. 1991).
[7] *See* MPEP § 2131.
[8] *Verdegaal Bros. v. Union Oil Co. of Calif.*, 2 USPQ2d 1051, 1053 (Fed. Cir. 1987).
[9] *Richardson v. Suzuki Motor Co.*, 9 USPQ2d 1913, 1920 (Fed. Cir. 1989).
[10] *Pac-Tex, Inc. v. Amerace Corp.*, 14 USPQ2d 187 (Fed. Cir. 1990).

The disclosed and claimed invention uses program analysis information obtained from analysis of a non-executing software program. The types of conventional program analysis information are known as being obtained from a non-executing software program. Under the applicable case law for anticipation, it is not enough that the Examiner find a reference which discloses something in a related technical field, i.e., computer software analysis.

Wygodny et al. facilitates the process of tracing the program execution paths as the program executes, and relates to the process of monitoring and analyzing the execution of computer programs during the debugging process. In Wygodny et al., the tracing is performed without requiring modifications to the executable modules or source code, and the trace data is collected according to instructions in a trace control dataset. Wygodny et al. propose a software system that facilitates the process of identifying and isolating bugs within a client program by allowing a developer to trace the execution paths of the client.

Generally, a debugger examines how variable values change as a program executes. The program analysis information of the disclosed and claimed invention, such as syntactic analysis tree, call graph, and flow graph, is statically obtained from source code, but is not generated during execution of the program.

The trace data as in Wygodny indicates values of executed functions and variable values when the program is executed. The volume of trace data may become large if the time required for executing programs is long, or if a large number of instructions are executed. Further, the trace data of Wygodny et al. is trace information generated by a debugger, and generated concurrent with the execution of the program.

A "trace" is defined as a debugging aid that chronicles the actions and results of individual steps in a program. A trace takes a user's program and places it under control of a special routine which monitors the progress of the program. Continuous execution of the user's program is replaced by a process whereby the trace program intercedes between steps of the user's program, displaying a variety of material before permitting execution of the next step. The contents of most types of traces are characterized by such items as a copy of the instruction, its location, and operand and register values before and after execution. Some trace facilities are concerned with the sequence of events in a program, as well as with the history of various data

items, so that indications may be included as to whether certain branches have been followed, or information about cyclic processes, as the processes are being actually executed.[11]

As for the specific deficiencies of the applied art with respect to the claimed invention, Wygodny et al. does not disclose a software analysis apparatus which includes, among other features, "...program analysis information generation means for automatically generating program analysis information required for analyzing a computer program; program analysis information storage means for classifying the program analysis information generated by said program analysis information generation means in an arbitrary unit or at an arbitrary timing, and sequentially storing the program analysis information in a predetermined data recording medium...", as recited in independent claim 1.

Wygodny et al. not only does not disclose generation of program analysis information, as that term would be understood by a reading of Appellants' disclosure, but Wygodny et al. also does not disclose program analysis information storage means for storing and classifying the program analysis information.

In the Examiner's response to previously submitted arguments, the Examiner asserts, without offering any evidence for the record, that "merely gathering and storing information to processes related to [known information types]...does not constitute the basis for that which is unique or novel",[12] and further that "it would have been obvious to classify syntactic data as such, or symbol data as such...but this would not be unique or novel."[13]

There is no evidence in the record to support these allegations of lack of novelty or of obviousness. Further, the rejections on appeal are for anticipation under 35 U.S.C. §102, and not for unpatentability under §103.

Accordingly, as the applied art does not disclose all the claimed features, the Examiner has not met his evidentiary burden for anticipation, and reversal of the rejection of independent claim 1, and dependent claims 2-6, 8-10, and 35 by the Honorable Board is requested.

---

[11] Encyclopedia of Computer Science, p. 1427, Van Nostrand Reinhold Co., New York, 1976.
[12] *See* Final Official Action, p. 9, lines 13-19.
[13] *See* Final Official Action, p. 10, lines 9-10.

D. Has the Examiner established that every recited feature of claims 13-17, 19-21, and 36 in Group II is disclosed by Wygodny et al. (US 6,282,701), and therefore anticipated under 35 U.S.C. §102(e)?

The citations to relevant case law on anticipation are provided in paragraph C above.

Wygodny et al. does not disclose a software analysis method, which includes, among other features, "...classifying the program analysis information in an arbitrary unit or at an arbitrary timing; sequentially storing the program analysis information in a predetermined data recording medium; and executing program analysis by reading out the program analysis information from said data recording medium", as recited in independent claim 13.

The Examiner admits, in his response to arguments cited above, that Wygodny et al. is deficient with respect to disclosing classifying program analysis information, but asserts, without offering any evidence, that it would be "obvious" to classify data, and that it is not novel to store program analysis information, as discussed above.

There is no evidence in the record to support these allegations of lack of novelty or of obviousness. Further, Appellants repeat that the rejections on appeal are for anticipation under 35 U.S.C. §102, and not for unpatentability under §103.

Accordingly, as the applied art does not disclose all the claimed features, the Examiner has not met his evidentiary burden for anticipation, and reversal of the rejection of independent claim 13, and dependent claims 14-17, 19-21, and 36 by the Honorable Board is requested.

E. Has the Examiner established that every recited feature of claims 24-28, 30-32, and 37 in Group III is disclosed by Wygodny et al. (US 6,282,701), and therefore anticipated under 35 U.S.C. §102(e)?

The citations to relevant case law on anticipation are provided in paragraph C above.

Wygodny et al. does not disclose a computer readable recording medium recording a computer program for making a computer implement, among other functions, "...a program analysis information classification function of ***classifying*** the program analysis information generated by the program analysis information generation function in an arbitrary unit or at an arbitrary timing; a program analysis information storage function of sequentially ***storing*** the

classified program analysis information generated by the program analysis information classification function in a predetermined data recording medium; and a program analysis function of executing program analysis by ***reading*** out the classified program analysis information from said data recording medium", as recited in independent claim 24.

The Examiner admits, in his response to arguments cited above, that Wygodny et al. is deficient with respect to disclosing classifying program analysis information, but asserts, without offering any evidence, that it would be "obvious" to classify data, and that it is not novel to store program analysis information, as discussed above.

There is no evidence in the record to support these allegations of lack of novelty or of obviousness. Further, Appellants again reiterate that the rejections on appeal are for anticipation under 35 U.S.C. §102, and not for unpatentability under §103.

Accordingly, as the applied art does not disclose all the claimed features, the Examiner has not met his evidentiary burden for anticipation, and reversal of the rejection of independent claim 24, and dependent claims 25-28, 30-32, and 37 by the Honorable Board is requested.

F. Has the Examiner established that every recited feature of claims 7, 11, 12, 18, 22, 23, 29, 33, and 34 in Group IV is disclosed by Wygodny et al. (US 6,282,701), and therefore anticipated under 35 U.S.C. §102(e)?

The citations to relevant case law on anticipation are provided in paragraph C above.

A common concept of the claims of Group IV is that program analysis information is stored in a data recording medium as a database and, in at least one dependent claim (34), in an object-oriented database.

Wygodny et al. does not disclose a software analysis apparatus which includes, among other features, "...program analysis information storage means for classifying the program analysis information generated by said program analysis information generation means in an arbitrary unit or at an arbitrary timing, and sequentially storing the program analysis information in a predetermined data recording medium... wherein said program analysis information storage means stores the program analysis information in said data recording medium as a database", as recited in dependent claim 7.

The Examiner does not address any specific, asserted disclosure in anticipation of dependent claim 7.

Wygodny et al. does not disclose a software analysis apparatus for generating program analysis information required for analyzing a computer program, which includes, among other features, "...means for hierarchically registering the generated program analysis information in a database in units of analysis objectives; [and] means for implementing analysis of the hierarchically registered program analysis information by reading out the program analysis information already registered in a predetermined layer in correspondence with an analysis objective upon analyzing the computer program", as recited in independent claim 11.

The Examiner does not address any asserted disclosure of dependent claim 11, other than to erroneously assert that Figs. 3A, 3B, 5-7, and 13-14 disclose storing analyzed data in a database.

These figures really disclose, in Fig. 3A - an illustration of a typical main frame window provided by the system's trace analyzer module; Fig. 3B – an illustration of a typical main frame window showing multiple threads; Fig. 5 – a trace option window that allows a developer to select the functions to be traced and the information to be collected during the trace; Fig. 6 – a file page window that provides a hierarchical tree of trace objects listed according to hierarchical level; Fig. 7 – a class page window that provides a hierarchical tree of trace objects sorted by class; Fig. 13 – a flowchart which illustrates the process of attaching to (hooking) a running process; and Fig. 14 – a flowchart which illustrates the process of loading an executable file and attaching to (hooking) the program.

None of these figures offered by the Examiner disclose a database or database structure.

Wygodny et al. does not disclose a software analysis method, which includes, among other features, "...classifying the program analysis information in an arbitrary unit or at an arbitrary timing; sequentially storing the program analysis information in a predetermined data recording medium; and executing program analysis by reading out the program analysis information from said data recording medium...wherein the program analysis information

storage step includes the step of storing the program analysis information in said data recording medium as a database", as recited in dependent claim 18.

The Examiner does not address any specific, asserted disclosure in anticipation of dependent claim 18.

Wygodny et al. does not disclose a software analysis method for generating program analysis information required for analyzing a computer program, which includes, among other features, "...hierarchically registering the generated program analysis information in a database in units of analysis objectives...", as recited in independent claim 22.

The Examiner does not address any specific, asserted disclosure in anticipation of independent claim 22 which relates to use of a database.

Wygodny et al. does not disclose a computer readable recording medium recording a computer program for making a computer implement, among other functions, "...a program analysis information classification function of classifying the program analysis information generated by the program analysis information generation function in an arbitrary unit or at an arbitrary timing; a program analysis information storage function of sequentially storing the classified program analysis information generated by the program analysis information classification function in a predetermined data recording medium...wherein the program analysis information storage function stores the program analysis information in said data recording medium as a database", as recited in dependent claim 29.

The Examiner does not address any specific, asserted disclosure in anticipation of dependent claim 29.

Finally, Wygodny et al. does not disclose a computer readable storage medium recording a program for making a computer implement, among other functions, "...hierarchically registering the generated program analysis information in a database in units of analysis objectives...", as recited in independent claim 33.

The Examiner does not address any specific, asserted disclosure in anticipation of independent claim 33, other than to assert, apparently with respect to dependent claim 34, without

offering any evidence, that "[o]bject-oriented database techniques are old and well known methods of database construction."[14]

Appellants point out that there is no evidence in the record to support these allegations of "well-known" technology, and Wygodny et al certainly does not disclose this dependent claim feature of claim 34. Appellants again reiterate that the rejections on appeal are for anticipation under 35 U.S.C. §102, and not for unpatentability under §103. There has been no indication of any art of record, which is properly combinable with Wygodny et al. in an unpatentability rejection, and which teaches or suggests all the recited features.

In terms of reliance upon common sense or ("well known") common knowledge, in *Zurko*, the Federal Circuit has reiterated to the Board of Appeals the following:

> ...the deficiencies of the cited references cannot be remedied by the Board's general conclusions about what is "basic knowledge" or "common sense" to one of ordinary skill in the art...the Board contended that even if the cited UNIX and FILER2 references did not disclose a trusted path, "it is basic knowledge that communication in trusted environments is performed over trusted paths" and, moreover, verifying the trusted command in UNIX over a trusted path is "nothing more than good common sense." *Ex parte Zurko*, slip op. at 8.
>
> We cannot accept these findings by the Board. This assessment of basic knowledge and common sense was not based on any evidence in the record and, therefore, lacks substantial evidence support. As an administrative tribunal, the Board clearly has expertise in the subject matter over which it exercises jurisdiction. This expertise may provide sufficient support for conclusions as to peripheral issues. With respect to core factual findings in a determination of patentability, however, the Board cannot simply reach conclusions based on its own understanding or experience -- or on its assessment of what would be basic knowledge or common sense.
>
> Rather, the Board must point to some concrete evidence in the record in support of these findings. To hold otherwise would render the process of appellate review for substantial evidence on the record a meaningless exercise. *Baltimore & Ohio R.R. Co. v. Aderdeen & Rockfish R.R. Co.*, 393 U.S. 87, 91-92 (1968) (rejecting a determination of the Interstate Commerce Commission with no support in the record, noting that if the Court were to conclude otherwise "[the] requirement for administrative decisions based on substantial evidence and reasoned findings -- which alone make effective judicial review possible -- would become lost in the haze of so-called expertise".

[14] *See* Final Official Action at p. 6, lines 18-19.

Accordingly, we cannot accept the Board's unsupported assessment of the prior art.[15]

Accordingly, as the applied art does not disclose all the claimed features, the Examiner has not met his evidentiary burden for anticipation, and reversal of the rejection of claims 7, 11, 12, 18, 22, 23, 29, 33, and 34 by the Honorable Board is requested.

IX. CLAIMS INVOLVED IN THE APPEAL

A copy of the claims involved in the present appeal is attached hereto as Appendix A. As indicated above, the claims in Appendix A do include the amendments filed by Applicant on April 8, 2002.

Respectfully submitted,

By Larry J. Hume

Larry J. Hume
Registration No.: 44,163
CONNOLLY BOVE LODGE & HUTZ, LLP
1990 M Street, N.W., Suite 800
Washington, DC 20036-3425

(202) 331-7111
(202) 293-6229 (Fax)
Attorneys for Applicant

[15] *In re Zurko*, 258 F.3d 1379, 59 U.S.P.Q.2d 1693 (Fed. Cir. 2001).

APPENDIX A

Claims Involved in the Appeal of Application Serial No. 09/241,735

1. (Amended) A software analysis apparatus comprising:

program analysis information generation means for automatically generating program analysis information required for analyzing a computer program;

program analysis information storage means for classifying the program analysis information generated by said program analysis information generation means in an arbitrary unit or at an arbitrary timing, and sequentially storing the program analysis information in a predetermined data recording medium; and

program analysis means for executing program analysis by reading out the program analysis information from said data recording medium.

2. An apparatus according to claim 1, wherein said program analysis means reads out the program analysis information from said data recording medium, and executes program analysis by an interactive process with an operator.

3. An apparatus according to claim 1, wherein said program analysis means reads out the program analysis information from said data recording medium, and executes program analysis by a batch process.

4. An apparatus according to claim 3, wherein said program analysis means generates at least one of metrics information, redundancy information, data flow anomaly information, and maintenance document information by the batch process.

5. An apparatus according to claim 1, wherein said program analysis information generation means generates a plurality of kinds of program analysis information in turn, and said program analysis information storage means stores the program analysis information in said data recording medium every time each kind of program analysis information is generated.

6. An apparatus according to claim 5, further comprising range instruction means for instructing a range of the plurality of kinds of program analysis information to be generated.

7. An apparatus according to claim 1, wherein said program analysis information storage means stores the program analysis information in said data recording medium as a database.

8. An apparatus according to claim 1, wherein the program analysis information includes at least one of:

a syntactic analysis tree generated on the basis of source code of the computer program;

a symbol table indicating meanings of symbols used in source code of the computer program;

a call graph or flow graph generated on the basis of the syntactic analysis tree;

data flow information generated on the basis of the syntactic analysis tree, symbol table, flow graph, and call graph; and

a program dependence graph or module I/O information generated on the basis of the syntactic analysis tree, symbol table, flow graph, call graph, and data flow information.

9. An apparatus according to claim 8, wherein said program analysis information generation means generates the syntactic analysis tree and symbol table, call graph and flow graph, data flow information, and program dependence graph and module I/O information in the order listed.

10. An apparatus according to claim 9, wherein said program analysis information storage means stores each program analysis information in said data recording medium every time said program analysis information generation means generates one of the syntactic analysis tree and symbol table, call graph, flow graph, data flow information, program dependence graph, and module I/O information.

11. (Amended) A software analysis apparatus for generating program analysis information required for analyzing a computer program, comprising:

means for hierarchically registering the generated program analysis information in a database in units of analysis objectives;

means for implementing analysis of the hierarchically registered program analysis information by reading out the program analysis information already registered in a predetermined layer in correspondence with an analysis objective upon analyzing the computer program.

12. An apparatus according to claim 11, wherein the database is an object-oriented database.

13. (Amended) A software analysis method, comprising:

automatically generating program analysis information required for analyzing a computer program;

classifying the program analysis information in an arbitrary unit or at an arbitrary timing;

sequentially storing the program analysis information in a predetermined data recording medium; and

executing program analysis by reading out the program analysis information from said data recording medium.

14. A method according to claim 13, wherein the program analysis step includes the step of reading out the program analysis information from said data recording medium, and executing program analysis by an interactive process with an operator.

15. A method according to claim 13, wherein the program analysis step includes the step of reading out the program analysis information from said data recording medium, and executing program analysis by a batch process.

16. A method according to claim 15, wherein the program analysis step includes the step of generating at least one of metrics information, redundancy information, data flow anomaly information, and maintenance document information by the batch process.

17. A method according to claim 13, wherein the program analysis information generation step includes the step of generating a plurality of kinds of program analysis information in turn, and the program analysis information storage step includes the step of storing the program analysis information in said data recording medium every time each kind of program analysis information is generated.

18. A method according to claim 13, wherein the program analysis information storage step includes the step of storing the program analysis information in said data recording medium as a database.

19. A method according to claim 13, wherein the program analysis information includes at least one of:

a syntactic analysis tree generated on the basis of source code of the computer program;

a symbol table indicating meanings of symbols used in source code of the computer program;

a call graph or flow graph generated on the basis of the syntactic analysis tree;

data flow information generated on the basis of the syntactic analysis tree, symbol table, flow graph, and call graph; and

a program dependence graph or module I/O information generated on the basis of the syntactic analysis tree, symbol table, flow graph, call graph, and data flow information.

20. A method according to claim 19, wherein the program analysis information generation step includes the step of generating the syntactic analysis tree and symbol table, call graph and flow graph, data flow information, and program dependence graph and module I/O information in the order listed.

21. A method according to claim 20, wherein the program analysis information storage step includes the step of storing each program analysis information in said data recording medium every time one of the syntactic analysis tree and symbol table, call graph, flow graph, data flow information, program dependence graph, and module I/O information is generated in the program analysis information generation step.

22. (Amended) A software analysis method for generating program analysis information required for analyzing a computer program, comprising:

hierarchically registering the generated program analysis information in a database in units of analysis objectives, and

implementing analysis by reading out the program analysis information already registered in a predetermined layer in correspondence with an analysis objective upon analyzing the computer program.

23. (Amended) A method according to claim 22, wherein the database is an object-oriented database.

24. (Amended) A computer readable recording medium recording a computer program for making a computer implement:

a program analysis information generation function of automatically generating program analysis information required for analyzing a computer program;

a program analysis information classification function of classifying the program analysis information generated by the program analysis information generation function in an arbitrary unit or at an arbitrary timing;

a program analysis information storage function of sequentially storing the classified program analysis information generated by the program analysis information classification function in a predetermined data recording medium; and

a program analysis function of executing program analysis by reading out the classified program analysis information from said data recording medium.

25. A medium according to claim 24, wherein the program analysis function reads out the program analysis information from said data recording medium, and executes program analysis by an interactive process with an operator.

26. A medium according to claim 25, wherein the program analysis function reads out the program analysis information from said data recording medium, and executes program analysis by a batch process.

27. A medium according to claim 26, wherein the program analysis function generates at least one of metrics information, redundancy information, data flow anomaly information, and maintenance document information by the batch process.

28. A medium according to claim 24, wherein the program analysis information generation function generates a plurality of kinds of program analysis information in turn, and the program analysis information storage function stores the program analysis information in said data recording medium every time each kind of program analysis information is generated.

29. A medium according to claim 24, wherein the program analysis information storage function stores the program analysis information in said data recording medium as a database.

30. A medium according to claim 24, wherein the program analysis information includes at least one of:

a syntactic analysis tree generated on the basis of source code of the computer program;

a symbol table indicating meanings of symbols used in source code of the computer program;

a call graph or flow graph generated on the basis of the syntactic analysis tree;

data flow information generated on the basis of the syntactic analysis tree, symbol table, flow graph, and call graph; and

a program dependence graph or module I/O information generated on the basis of the syntactic analysis tree, symbol table, flow graph, call graph, and data flow information.

31. A medium according to claim 30, wherein the program analysis information generation function generates the syntactic analysis tree and symbol table, call graph and flow graph, data flow information, and program dependence graph and module I/O information in the order listed.

32. A medium according to claim 31, wherein the program analysis information storage function stores each program analysis information in said data recording medium every time the program analysis information generation function generates one of the syntactic analysis tree and symbol table, call graph, flow graph, data flow information, program dependence graph, and module I/O information.

33. (Amended) A computer readable storage medium recording a program for making a computer implement a function of:

generating program analysis information required for analyzing a computer program,

hierarchically registering the generated program analysis information in a database in units of analysis objectives, and

implementing analysis by reading out the program analysis information already registered in a predetermined layer in correspondence with an analysis objective upon analyzing the computer program.

34. A medium according to claim 33, wherein the database is an object-oriented database.

35. The apparatus of claim 1, wherein said program analysis means analyzes where a given line in a source code of the computer program has an influence on the source code.

36. The method of claim 13, wherein said automatically generating program analysis information includes analyzing where a given line in a source code of the computer program has an influence on an execution of the source code.

37. The computer readable recording medium of claim 24, wherein the program analysis function includes a function of determining where a given line in a source code of the computer program has an influence on an execution of the source code.